NPS52-85-012

NAVAL POSTGRADUATE SCHOOL

Monterey, California

WORKSTATION GRAPHICS CAPABILITIES FOR THE 1990'S AND BEYOND

MICHAEL J. ZYDA

SEPT 1985

Prepared for:

f of Naval Research
ngton, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R.H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

This report was prepared by:

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS52-85-012	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): WORKSTATION GRAPHICS CAPABILITIES FOR THE 1990's AND BEYOND	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Michael J. Zyda	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943-5100	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61152N: RR000-01-NP, N0001485WR41005
11. CONTROLLING OFFICE NAME AND ADDRESS: Chief of Naval Research, Arlington, VA 22217	12. REPORT DATE: Sept 1985
	13. NUMBER OF PAGES: 35
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Algorithms, architecture, contour surface display generation, real-time display generation, graphics workstations

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We present in this paper a look at the future graphics capabilities of the workstation. We begin by examining the cyccles of special hardware development that have occurred for graphics systems in general. We show how the current evolution of the graphics workstation is a direct response to applications user desires for higher performance, graphics systems. The software and hardware levels that perform the input and output graphics operations for the workstation are described with an eye towards categorizing future graphics capabi-

lities. The implementation of those levels in the Silicon Graphics, Inc. IRIS is cited as an example of the leading edge for graphics capabilities in a workstation. Current research leading to future enhancements of the graphics workstation is presented as a continuation of the historical response to applications user desires for ever higher performance, interactive systems.

Workstation Graphics Capabilities for the 1990's and Beyond ‡

Michael J. Zyda

Naval Postgraduate School,

Code 52, Dept. of Computer Science,

Monterey, California 93943

ABSTRACT

We present in this paper a look at the future graphics capabilities of the workstation. We begin by examining the cycles of special hardware development that have occurred for graphics systems in general. We show how the current evolution of the graphics workstation is a direct response to applications user desires for higher performance, graphics systems. The software and hardware levels that perform the input and output graphics operations for the workstation are described with an eye towards categorizing future graphics capabilities. The implementation of those levels in the Silicon Graphics, Inc. IRIS is cited as an example of the leading edge for graphics capabilities in a workstation. Current research leading to future enhancements of the graphics workstation is presented as a continuation of the historical response to applications user desires for ever higher performance, interactive systems.

Categories and Subject Descriptors: I.3.1 [**Hardware Architecture**]: architectures, parallel processing, VLSI implementations; I.3.2 [**Graphics Systems**]: multiprocessing systems; I.3.3 [**Picture/Image Generation**]: surface visualization; I.3.6 [**Methodology and Techniques**]: contouring, interactive systems, parallel processing; I.3.7 [**Three-Dimensional Graphics and Realism**]: line drawings, line generation algorithms, real-time graphics, surface plotting, surface visualization, surfaces; I.3.m [**Miscellaneous**]: VLSI;

General Terms: Algorithms, architecture;

Additional Key Words and Phrases: contour surface display generation, real-time display generation, graphics workstations;

‡ This work has been supported by the NPS Foundation Research Program. The work was originally presented at an Institute for Graphic Communication conference on Engineering Workstations, the 3rd of February 1985.

1. Introduction

Graphics workstations represent the culmination of a long history of hardware developments for purposes of real-time, interactive applications systems. The idea behind such systems is to provide the human user immediate feedback of visual information in response to any physical control manipulations made. Such capabilities are an integral part of visual training simulators, command/control situations, and other time-critical applications. Historically, the effort to improve the capabilities of such systems has been a push-and-pull cycle of increasing applications user demands driving special hardware additions to the graphics system. In order to understand the future capabilities of such systems, we must examine the cycles of hardware development.

In the early days of computer graphics, applications users were happy if they could just get a picture to the display device. It did not to matter that the display device took two to three minutes for one picture as the alternative was to not be able to get the particular application done. In those early days, the computer was generally a single user system, with the graphics applications program consuming all available resources. The key problem with respect to interactive systems was that there was a lot of idle user time during the waits for the next display. Consequently, one of the first problems that was solved with special hardware was the speeding up of picture delivery. This can be considered the first cycle of special hardware for the graphics system.

Applications users readily took to computer graphics once they saw that they could get their picture to the display device in a reasonable amount of time. In fact, applications users took to computer graphics with such a fervor that they began demanding what to them seemed like the next logical development, the addition of matrix multipliers for the real-time operations necessary for rotating, scaling, and translating vectors. This was the second cycle of special hardware additions to the graphics system. This addition to the display system was quite important in that it allowed the development of real-time interactive applications not previously possible without the special hardware. (One example of this has been the near abandonment in the field of chemistry of the use of hard models of large molecules for the more readily manipulated computer models).

Anyone who has spent any amount of time with applications users knows quite well that they are never completely satisfied. The addition of the special hardware for matrix multipliers came towards the end of the cycle of single user minicomputer systems. Applications programmers momentarily got used to the immediate response of the single user computer graphics system, and then almost immediately lost that capability. This capability was lost due to the simple fact that the applications users outgrew the single user minicomputer systems, and moved onto the larger, shared super-minicomputers. The third cycle of improvements to the graphics system was in response to that loss. This cycle is typified by the offloading of the graphics and interaction functionalities from the host computer to a special processor dedicated to the graphics system. The goal behind this was to reclaim the real-time, interactive capabilities lost during the move to the shared super-minicomputer. This cycle created the modern interactive graphics workstation.

2. Interactive Graphics Workstation Organization

Current high performance graphics workstations have some variant of the organization depicted in Figure 1. In that figure, we see a central bus, typically the IEEE Multibus, off of which hang the CPU, the terminals, the disk drives, the Ethernet interfaces, and the other miscellaneous output devices. On the other side of the CPU, we see a bus going to a unit labeled DPU, or display processing unit, with that bus passing through and towards the actual display device, or display surface. Connected to the DPU are an array of interactive devices, i.e. mouse devices, joysticks, dials, buttons, switches, data tablets, light pens, and perhaps, a keyboard. For this study, we are primarily interested in the part of Figure 1 directly concerned with graphics.

With respect to high performance graphics, and the top half of Figure 1, there are two operations with which we are concerned: *(1) Getting the Picture There* (from the applications program to the display surface), and *(2) Manipulating the Picture* (by way of some movement of the interactive devices such that a picture change is generated). The first operation, *Getting the Picture There,* is most often termed the "output" function in the field of computer graphics. This means that we use some mathematical description encoded in the applications program to put a

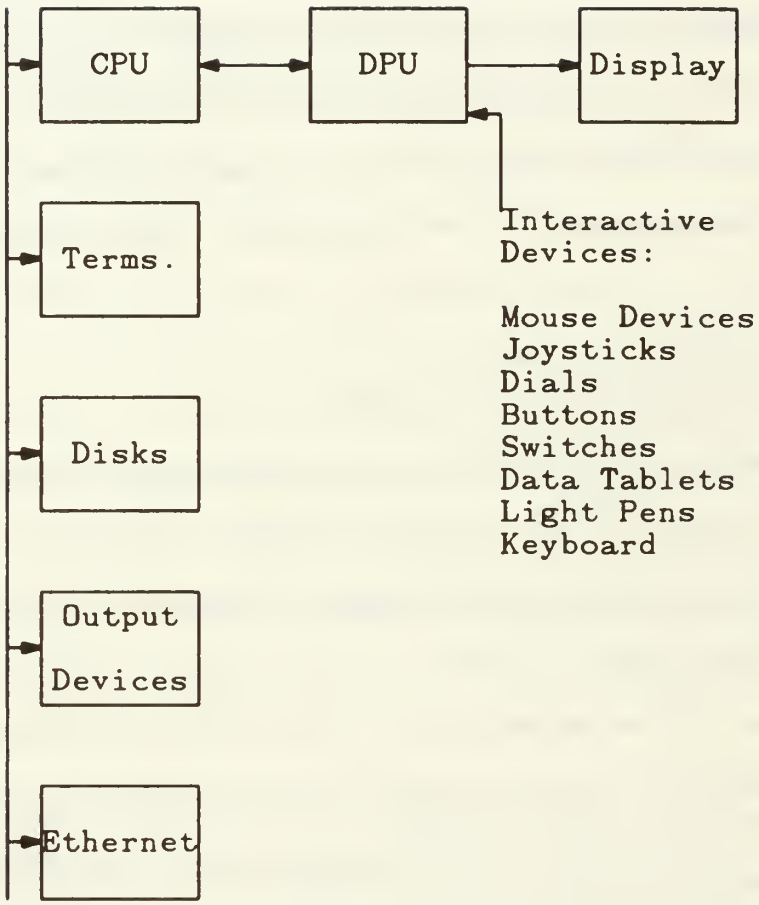


Figure 1
Basic Block Diagram of a Typical Interactive Graphics Workstation

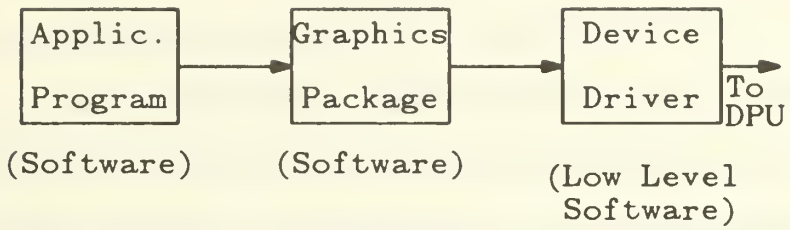


Figure 2
Software for the Output Function

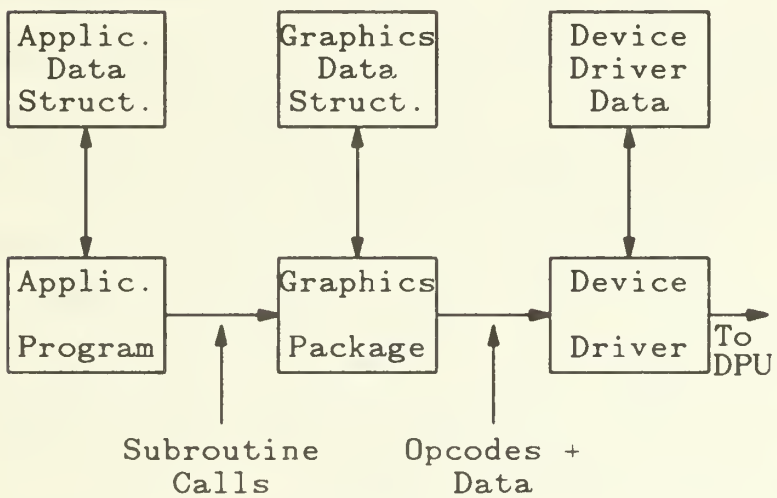


Figure 3
A Look at the Applications Program

visual display, or output, on the display surface. In order to properly understand the output function, we need to examine both the software and the hardware currently used to perform that function.

2.1. Software for the Output Function

A sketch of the levels of software involved in performing the output function is seen in Figure 2. In that figure, we see an applications program (software) making calls to a graphics package (software), with those calls being converted into calls to a device driver (low level software). Beyond the device driver are even lower level software calls, or perhaps, commands directed to the DPU's hardware.

The applications program is the start of the pathway to the DPU. The applications program is the set of computer instructions that maintain the abstract mathematical description, or model, of the applications user's world. This means that if the applications program is a VLSI design program, that it is the set of instructions that knows about transistors, registers, etc.. The applications program makes calls to the graphics package (Figure 3). The graphics package makes some transformations on the data passed to it, and passes the transformed data onto the device driver. Part of this transformation step is putting the data into an opcode format the device driver expects. The final operations in the software pathway to the DPU are performed by the device driver. The device driver converts the data received into the opcode streams required by the DPU. The next step in the output function is a hardware step. i.e. the DPU's conversion of that stream into a form that can be sent to the display.

2.2. Hardware for the Output Function

Once we have a rough idea of the software pathway necessary to perform the output function, we then need to look at the hardware pathway. The hardware pathway is mostly contained within the DPU (Figure 4). The only part of the hardware pathway that is outside of the DPU is the pathway from the refresh subsystem to the display surface. The DPU is comprised of the following pieces of hardware: the display controller, the raster subsystem, the frame buffer, and the

refresh subsystem. We can best understand the function of these different components of the DPU if we discuss them in terms of their data flow. At the start of Figure 4, we see an opcode stream entering the display controller. This stream contains the instructions and data output by the device driver software. The data that leaves the the display controller for the raster subsystem are lines and polygons, and their associated colors and fills. The raster subsystem, in turn, converts those lines and polygons into the set of pixels necessary for their representation in the frame buffer. The frame buffer's pixels are read by the refresh subsystem, which converts those pixels into electron beam deflections. With the above brief overview of the data flow of the DPU in mind, we can define the parts of the DPU with respect to the graphics capabilities needed for the output function. We begin by looking in more detail at the display controller.

2.2.1. Graphics Capabilities for the Output Function: Display Controller

The display controller is best understood in terms of its data flow, and its operational capabilities. As seen in Figure 4, the display controller has an opcode stream coming in, as formatted by the device driver, and has vectors and polygons going out. The stream coming in is comprised of opcodes followed by data. The data is a collection of untransformed coordinates, matricies, text, colors, linestyles, fills, etc.. The data going out from the display controller is comprised of transformed coordinates in frame buffer space, text, colors, linestyles, fills, etc..

The operations the display controller performs on the input data are the following: (1) matrix transformations, i.e. rotations, scalings, translations, (2) coordinate system mappings, and clippings, i.e. world coordinates to frame buffer coordinates, (3) projections, i.e. 3D to 2D, perspective/orthographic, and (4) display list management. Now the first three operations are familiar to those with a background in graphics. The only one that requires some explanation is the fourth operation, that of display list management. A display list is a set of instructions describing the desired image. In reference to the previous discussion, it is the data input as an opcode stream. The display list is interpreted by the display controller. The display controller determines the operations it needs to perform on the input data from the display list, and passes on the remainder of the work to the next system in the hardware path, the raster subsystem.

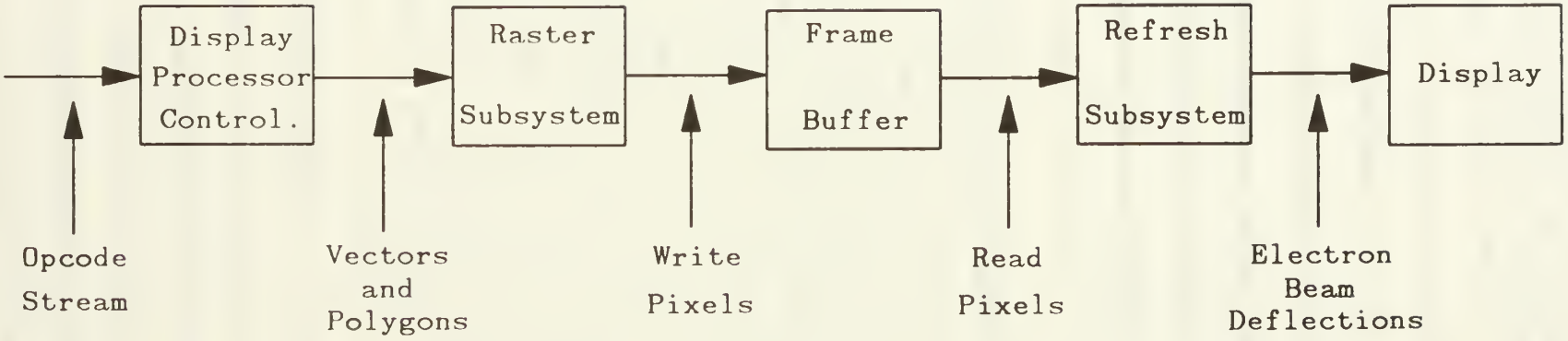


Figure 4

Hardware Path for the Output Operation: Display Processing Unit

2.2.2. Graphics Capabilities for the Output Function: Raster Subsystem

The raster subsystem receives lines and polygons that have been transformed into frame buffer space, text, colors, linestyles, and polygon fillstyles from the display controller. Before we can discuss the operations performed by the raster subsystem, we must first describe the frame buffer, the destination for the output from the raster subsystem.

The frame buffer is a two-dimensional array of memory. Each position in the frame buffer has a value, called a pixel. The data at each pixel location corresponds to the color that should be drawn at that position on the graphics display. The operations performed by the raster subsystem are all destined for output to the frame buffer. The raster subsystem converts input line segments into the set of pixels necessary for the display of those segments. The raster subsystem also converts input polygons to the set of pixels necessary for the display of their boundaries, and interior fills. The raster subsystem provides a similar treatment for text, i.e. it fills the frame buffer with the appropriate patterns of pixels.

2.2.3. Graphics Capabilities for the Output Function: The Refresh Subsystem

The final part of the hardware pathway for the output function is the refresh subsystem. The refresh subsystem reads rows of pixels from the frame buffer and produces the necessary electron beam deflections on the cathode ray tube of the graphics display. It performs this operation either every sixtieth or every thirtieth of a second, depending on the cathode ray tube driver technology used.

2.3. Software and Hardware for the Graphics Input Function

With respect to the graphics workstation, the second operation with which we are concerned is the input function (Figure 5). It is more correctly termed the picture manipulation function but we stick to the accepted terminology. It is called the input function because the operation the applications program is performing is reading a value from an interactive device. The input function is really a feedback function, i.e. we read some control values from the interactive devices at the DPU, pass those values back to the applications program, make a change in the

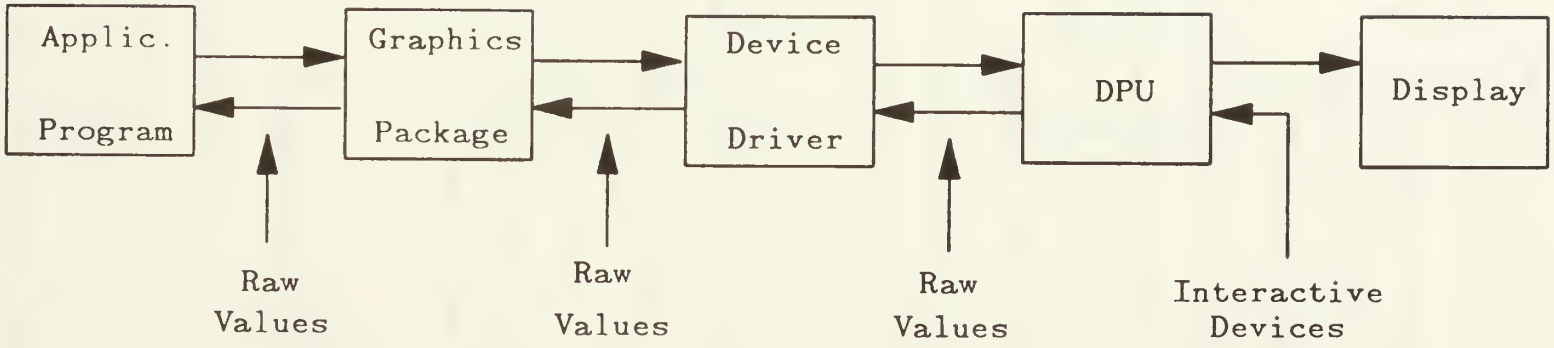


Figure 5

Organization of the Input Function: "Manipulating the Picture"

picture at that level, and then send the new picture via the output pathway described above. This is emphasized in Figure 5 by a pathway of directed arrows from the DPU in the direction of the applications program, and another set of directed arrows back towards the DPU and then the display. The values read from the interactive devices are typically passed back to the applications program in an unchanged, or raw form. In the applications program, the raw values are utilized in an applications programmer written procedure to modify some aspect of the current display. An example of this is the conversion of a dial value into an angle, with that angle being plugged into a rotation matrix, or rotation command passed back to the DPU.

2.3.1. Hardware for the Input Function

Other than the actual interactive device hardware, and the interfaces to support those devices, there tends not to be much special hardware to support the input function. There are two major exceptions: (1) direct cursor movement hardware support, and (2) display list parameter modification hardware. Cursor devices, i.e. mouse devices, data tablet pens, and light pens, sometimes have hardware support that eliminates the need to feed raw data values back to the applications program to change the position of the cursor. This operation is generally carried out by the DPU. It does not require much in the way of special hardware.

Display list parameter modification hardware is similar in goal to that of the hardware for direct cursor movement. This hardware provides a mechanism by which modifications of parameters embedded in display lists can be routed directly from the DPU on interactive control movement, rather than through the longer applications program loop. An example of this type of operation is the routing of a raw control value, or some simple linear modification of that raw value, as a direct replacement of an argument in an instruction in a display list, i.e. replacing the angle value in a rotation command. This type of operation is similar in concept to the ill-thought-of practice of self-modifying code and requires some knowledge of the internal structure of the selected graphics system's display lists.

Besides hardware for the above two input operations, there is generally no special hardware for the input function. This lack of hardware limits the sophistication of the types of interactive

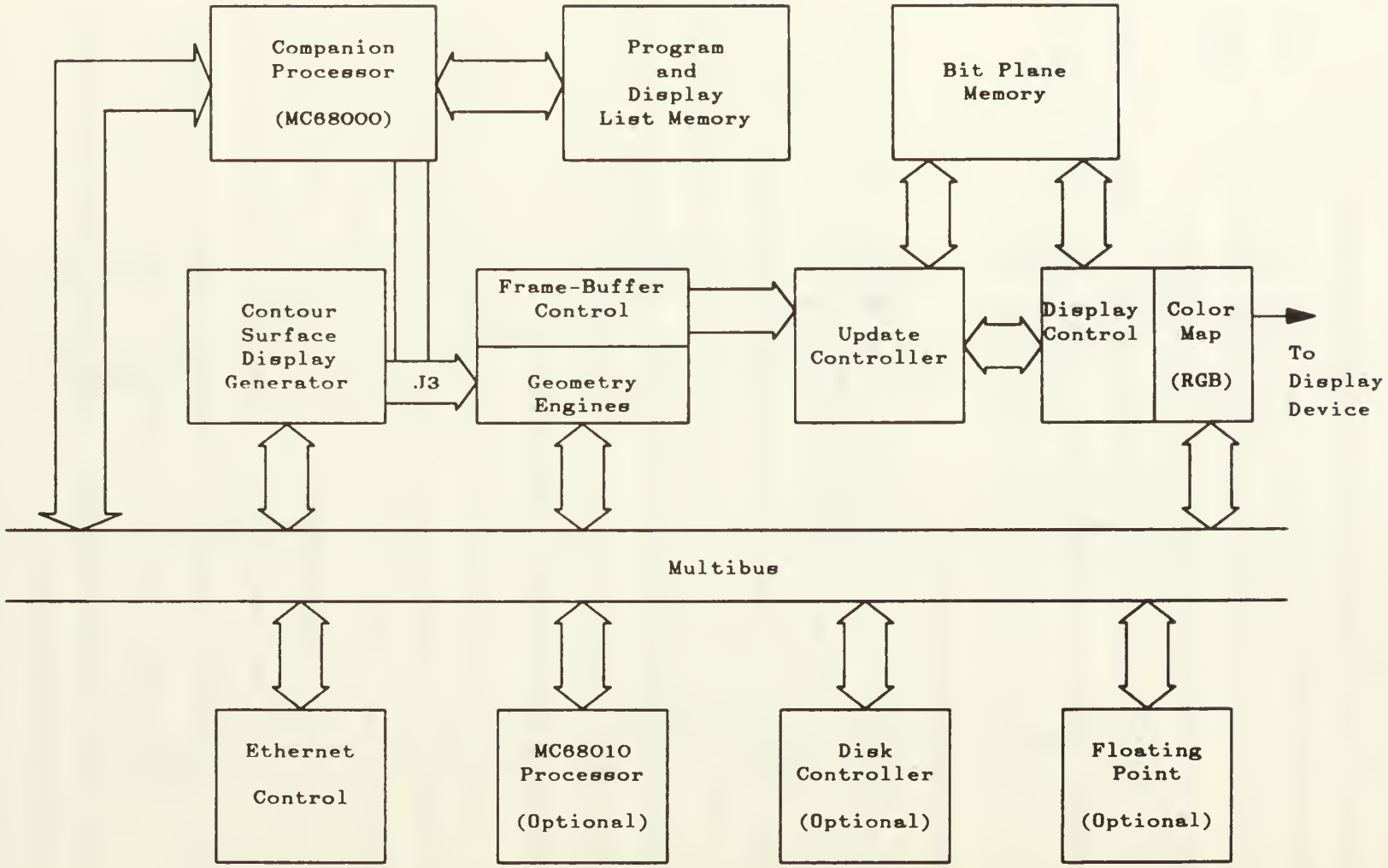


Figure 6
Block Diagram of the Silicon Graphics, Inc. IRIS
(with additional, non-SGI Contour Surface Display Generator)

operations we are currently capable of performing in the graphics workstation. Without special hardware to support special input functions, we are limited to the slow, feedback pathway from the DPU to the applications program and back.

3. Leading Edge Graphics Workstation Capabilities

To this point we have not talked about commercially available graphics workstations' capabilities. We have only given a generic description of the input and output functions with respect to the hardware and software subsystems necessary to perform those functions. To show the leading edge of technology for the graphics workstation, we refer back to some of those descriptions and point out how they are available on one high-performance graphics workstation.

3.1. The Silicon Graphics, Inc. IRIS Workstation

In the section on graphics capabilities for the output function of the display controller, we listed the operations that are performed by that part of the DPU. They are (1) matrix transformations, (2) coordinate system mappings, and clippings, (3) projections, and (4) display list management. One of the leading edge developments that have been accomplished for this subsystem of the DPU is the addition of a special pipeline processor to perform the first three functions of the list. The best example of this for a graphics workstation is that of the Silicon Graphics, Inc. IRIS (Figures 6 and 7, and [2]). The IRIS system has a "Geometry Pipeline" for these operations. This pipeline has five major components, all implemented via special purpose VLSI chips. The first component, as shown in Figure 7, is a special VLSI subsystem to convert world, or applications program coordinates to Geometry Engine floating point format. The second component is a four chip pipeline for matrix multiplication. This part of the pipeline operates on 4 x 4 matricies setup for rotations, translations, and scalings. The third component is a six chip pipeline of clippers that perform geometric clipping, i.e. top, bottom, left, right, near, and far clipping. The fourth component is a two chip pipeline, labeled scalers, that performs a perspective division, the projection operation, and the mapping of the three-dimensional coordinates to two-dimensional space. The final component of the pipeline is a VLSI subsystem to convert back from Geometry

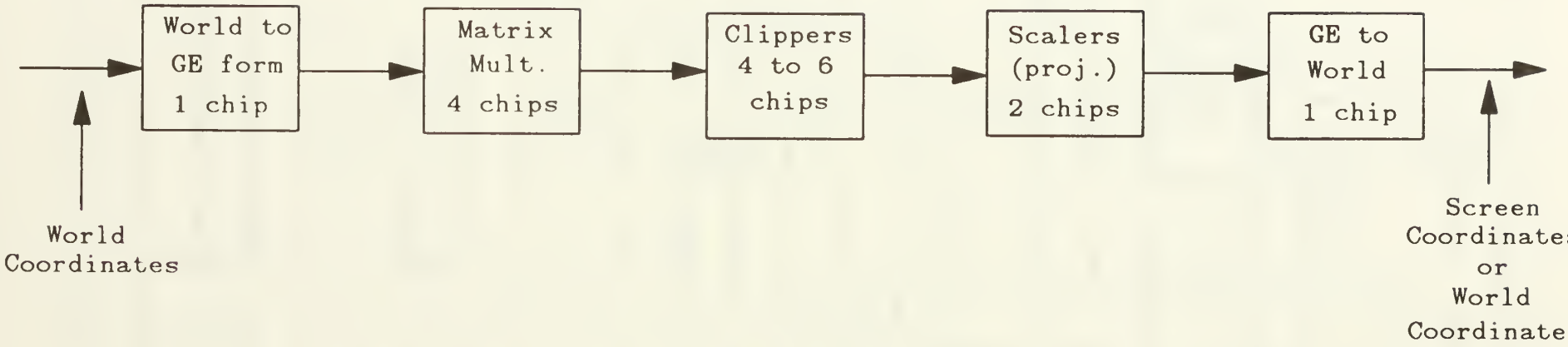


Figure 7
Silicon Graphics, Inc. IRIS Geometry Pipeline

Engine floating point format to world coordinate format. We note here that this operation, though not directly useful for the graphics output operation, is somewhat useful when utilizing the Geometry Engine pipeline as a computational engine. The brochure produced by Silicon Graphics, Inc. for the IRIS-2400 model containing this pipeline cites a capability for 80K 4 x 4 transformations per second.

The Silicon Graphics, Inc. IRIS-2400 has other leading edge functionalities present in special hardware. One of these is in the raster subsystem of the DPU, polygon fill hardware. This hardware converts two and three-dimensional polygon data into the set of textured and colored pixels that represent the polygon in the frame buffer. The rate cited for the IRIS-2400 is the capability for filling polygons at approximately 44 million pixels per second.

Another leading edge function of the IRIS is depth cueing hardware. Depth cueing is the intensity modulation of line segments so that components of the segment near the viewer appear brighter, and those farther away appear dim. The rate cited for this hardware capability is from 1.5 to 3 million pixels per second.

Gouraud shading is another feature of the IRIS-2400. Gouraud shading is a smooth shading algorithm useful in depicting surfaces via computer graphics. This algorithm works by taking the polygons that form the surface and shading those polygons by linear interpolation of the color intensities specified at the verticies of the polygon. This technique eliminates intensity discontinuities and produces a smoother, more realistic surface. The rate cited for this hardware capability is up to 3 million pixels per second.

Hidden surface elimination is provided via special hardware on the IRIS-2400. The hardware addition, called a Z-buffer, is a special piece of memory the same two-dimensional size as the frame buffer. Depth information, z coordinate values, is stored into this memory at the same time as color information is written into the frame buffer. This means that for each pixel in the frame buffer, there is a matching z coordinate. This information is used in the following fashion. As each new piece of the picture is processed by the raster subsystem, the pixel values are compared against those already in the Z-buffer. If the new pixel is closer, the color associated

with that pixel replaces the old one in the frame buffer, and the new z coordinate is written into the Z-buffer. If the new pixel is farther away than that indicated in the Z-buffer, the pixel is discarded. This special hardware addition is to the raster subsystem of the IRIS. Though no value is cited in the IRIS literature for the speed of this Z-buffering technique, it should operate at approximately the same rate as the polygon fill hardware, with some degradation due to the additional z coordinate value that needs to be propagated.

4. Trends in Graphics Capabilities for the Workstation

The above is a quick overview of one leading edge graphics workstation. There are others that exhibit similar capabilities, though most are not nearly the speed of the IRIS. From this brief look at the leading edge though, we see two trends, (1) the increasing importance of high performance graphics functionality in the workstation, and (2) the increasing use of VLSI technology to implement this functionality. The first trend, the increasing importance of high performance graphics functionality, is a continuation of the cycles of hardware additions requested by the ever unsatisfied graphics applications user. We do not expect this trend to diminish. In the past, the graphics applications user became accustomed to new hardware additions rapidly, only to turn around almost immediately with new requests. Given human nature, we do not expect this to change.

To understand the second trend, the increasing use of the VLSI technology to enhance the graphics capabilities of the workstation, we need to answer the question, what does VLSI provide? VLSI provides the capability for the parallel operation of large numbers of relatively inexpensive processors [8, 11]. Currently, we see 2 million transistors per chip in the research laboratory [9]. We are promised 10 million transistors per chip sometime between the years 1990 and 2001 [12]. From these numbers, graphics researchers tend to see tens of processors per chip, all operating in parallel on some graphics algorithm ‡.

‡Some landmarks for transistor count are 68,000 transistors in the Motorola MC68000 (18,000 in the processor, 50,000 in the PLA and ROM) [4], 18,000 transistors in the Z8000 [4], 40,000 transistors in one Geometry Engine chip [3], and 194,000 transistors in the Motorola MC68020 [7].

4.1. Fourth Cycle of Hardware Improvements to the Graphics System: Research

Besides the improvement of the capabilities of the standard hardware that performs the input and output functions of the graphics workstation, we see the start of a fourth cycle of special hardware developments also utilizing the VLSI technology. In this new cycle, the prominent work is the design of special, application dependent VLSI architectures for the real-time display generation of select graphics algorithms. The thrust of this research is the development of a methodology for taking a graphics algorithm and producing a silicon system that performs that algorithm. (The need for a methodology is quite simply to save time for the next algorithm through the hardware development process.) The scope of this work is quite large in comparison to the other cycles of special graphics hardware development. It encompasses the areas of real-time graphics software engineering, and VLSI computer architectures. Real-time graphics software engineering is part of this effort in that before one commits to implementing a particular graphics algorithm in silicon, one needs to be able to evaluate whether or not that algorithm can be computed in real-time on a currently available, high-performance graphics system. The research effort is to produce a system that can automatically model the desired algorithm such that runtime parameters can be obtained for hypothetical architectures, i.e. known processors like the MC68000 for subparts of the larger algorithm.

VLSI computer architectures are part of this effort in that the hypothetical architectures modeled are those capable of being implemented in VLSI. The research effort is twofold. The first part is the determination and evaluation of a special architecture for the studied algorithm. The determination of the architecture is accomplished through iterative design refinement driven by previous experience with such special processors. The evaluation of the architecture is both a runtime evaluation, and a technological evaluation. The runtime evaluation determines if the studied algorithm is capable of being executed in real-time on the hypothetical architecture. The technological evaluation determines if the proposed architecture is capable of being built within current technological constraints. Part of this effort is the examination of the changes required in the design of the graphics system that receives the output of the real-time display generator.

The second part of the research effort in the area of VLSI computer architectures is the evaluation and refinement of the software tools available for putting an architecture on silicon. Since VLSI technology is relatively new, the available software tools for producing special purpose VLSI chips and systems are crude. The research of the fourth cycle presupposes the existence of such software. Since this is clearly not the case, this research effort necessarily encompasses the refinement and development of such software tools.

4.2. Where We Are Today in the Fourth Cycle

The initial special hardware efforts of the fourth cycle are the construction of single board VLSI multiprocessors compatible with commercially available, high-performance graphics workstations. The selection of the commercially available workstation as the bed for the special hardware additions cuts the research effort with respect to real-time display generators in half, by delaying for later consideration possible changes to the design of the display system ‡. Such a sectioning of the research effort allows the design and testing of single board parts of perhaps much larger VLSI systems. One such effort underway at the Naval Postgraduate School is the design of a Multibus compatible, single board VLSI multiprocessor for generating contour surface displays in real-time [3].

4.2.1. Contour Surface Display Generator

The goal of the contour surface display generator is to produce and deliver to the display surface of a graphics workstation, in one-thirtieth of a second, the complete contour surface display generated from a 30 x 30 x 30 three-dimensional grid. The application in mind for this system is one directly from X-ray crystallography, the determination of molecular structures from electron density data [1]. Such an operation is executed interactively by using a computer graphics program that displays a Dreiding (stick) model of the molecule, inside a contour surface display of the corresponding region of the molecule's electron density grid. In addition to the graphics function, the computer program monitors a series of signals generated by the user, while

‡ There are currently substantial research efforts in the direction of the redesign of the graphics system [5, 10].

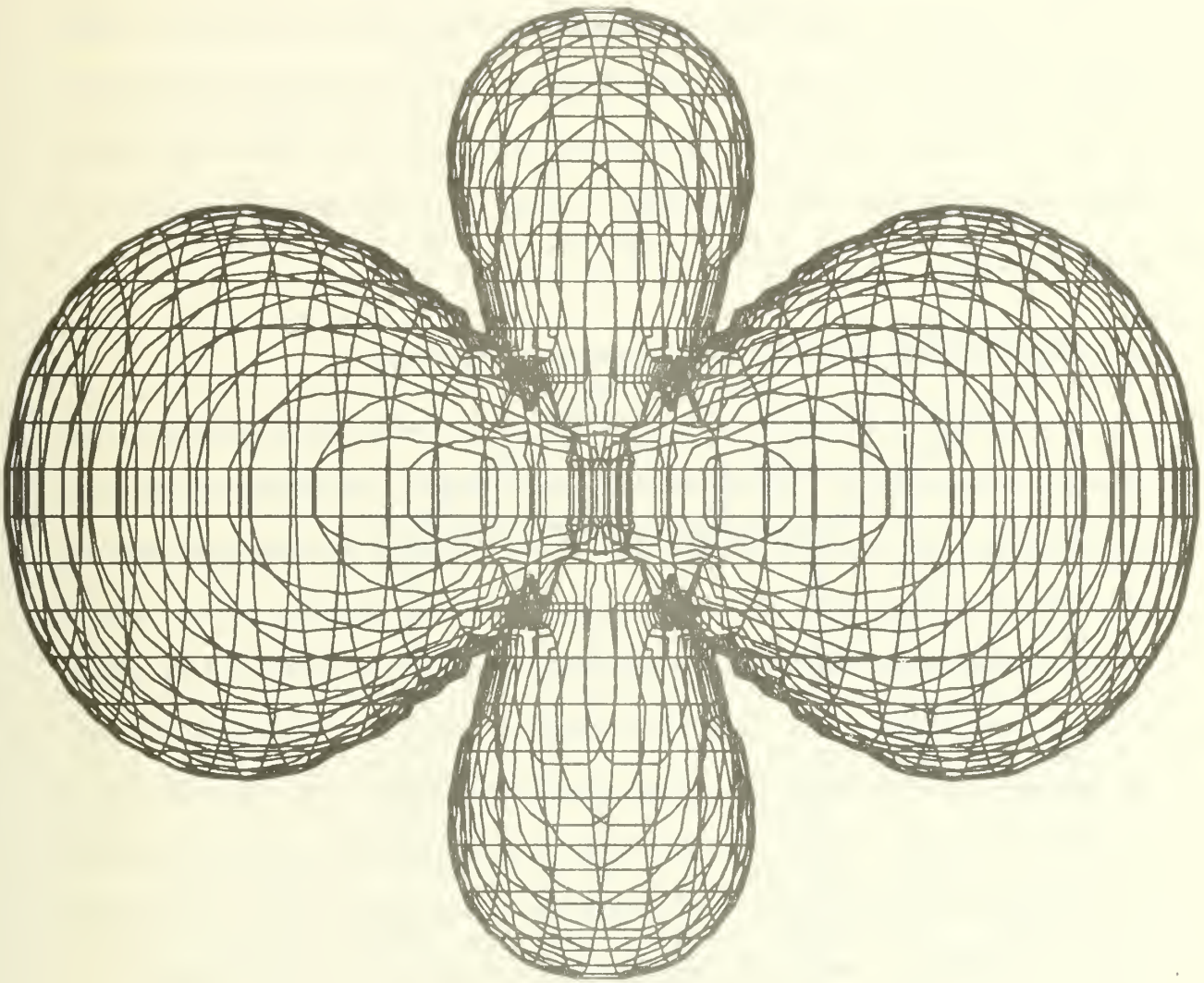
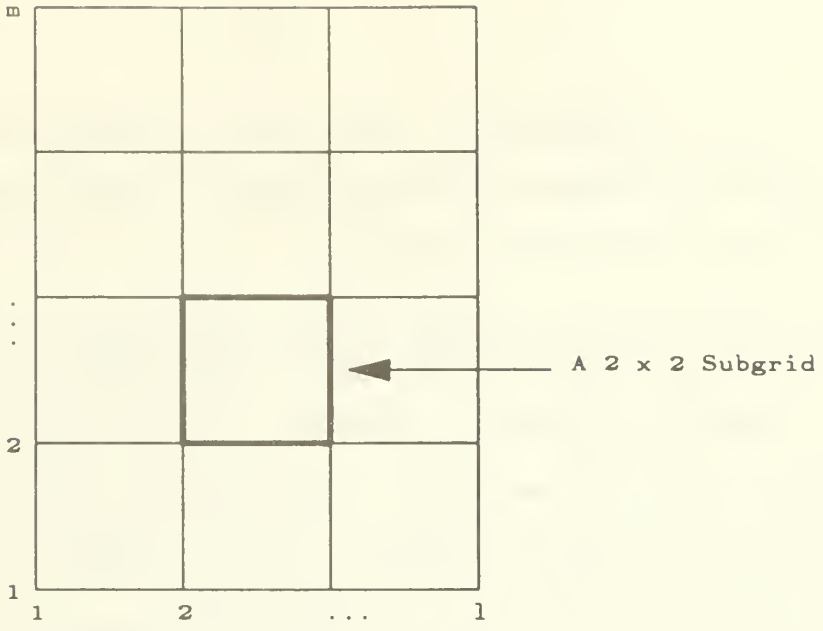


Figure 8
Contour Surface Display Generated from a Hydrogen Atom
Wavefunction Squared (3dz2 orbital)

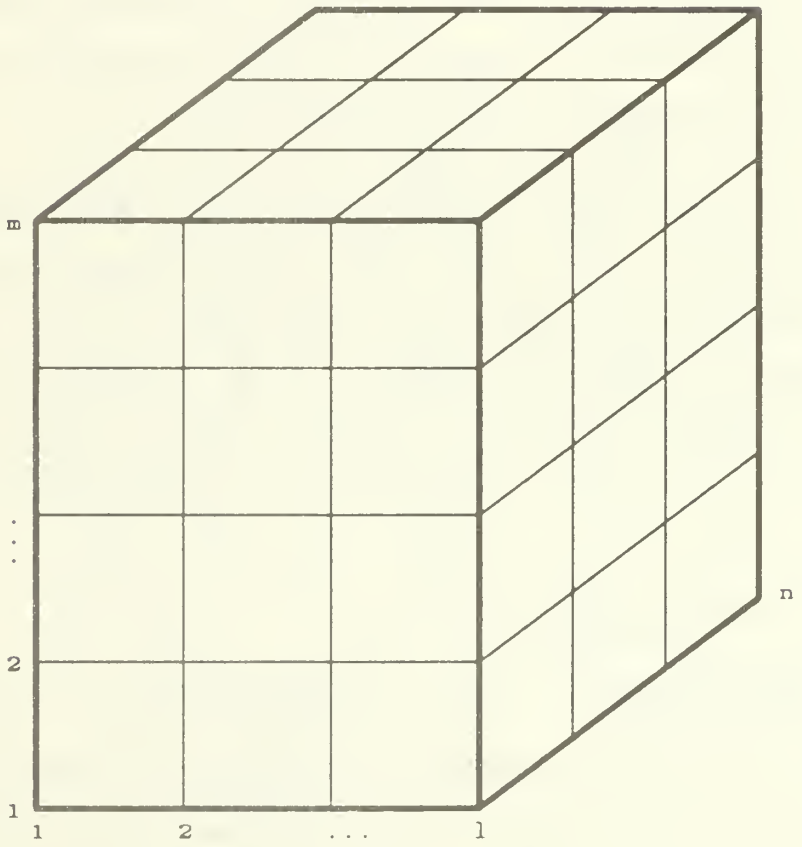
the user is turning the various knobs on a control console [16]. The values read from these knobs are interpreted by the program as modifications to either the molecule or the surface display. Modifications to the molecule take the form of bond rotations or bond lengthenings. Modifications to the contour surface display take the form of an increase or decrease of the contour level. The goal of this process is to produce the stick model of the molecule that best fits inside the given electron density data set. The user can determine whether or not the model fits the density grid by modifying the contour level, shrinking the contour surface to the molecule. Similarly, the user can expand the contour surface from the stick model for better visibility. This function requires that the hardware have the capability to rapidly change the contour display as its contour level changes.

4.2.2. Decomposable Algorithm for the Contouring Operation

The algorithm around which the design of the contour surface display generator is constructed is presented in [14]. That algorithm is constructed from a two-dimensional contouring algorithm that is used to contour all the possible planar, orthogonal, two-dimensional grids of a larger three-dimensional grid. The two-dimensional contouring algorithm of that study is comprised of components, called algorithm components, that operate on individual 2 x 2 subgrids of a larger two-dimensional grid. (Note: a 2 x 2 subgrid is defined to be that portion of the two-dimensional grid bounded by four adjacent grid points.) In the algorithm, the computations necessary for generating the contour lines for a single 2 x 2 subgrid are independent from those required for any other 2 x 2 subgrid. If we compute the contours corresponding to contour level k for all 2 x 2 subgrids of a two-dimensional grid, then we will have determined the complete set of contours for that grid. If we compute the contours corresponding to contour level k for all possible 2 x 2 subgrids of the larger three-dimensional grid, then we will have the complete contour surface display for that grid. The assemblage of the contours created by this process, i.e. the simultaneous display of all the contours created for all 2 x 2 subgrids of the larger three-dimensional grid, produces a "chicken-wire-like" contour surface display (Figure 8). The full development of this algorithm can be found in [14]. We refer to the results of those studies, and



A 2D Grid of Size l x m Has
(l - 1) x (m - 1) 2 x 2 Subgrids.



A 3D Grid of Size l x m x n Has
l x (m - 1) x (n - 1) + m x (l - 1) x (n - 1) + n x (l - 1) x (m - 1)
2 x 2 Subgrids

Figure 9
2 x 2 Subgrid Count for 2D and 3D Grids

do not cover the algorithm here in great detail. We only note that for the largest three-dimensional grid of interest for the above application, a 30 x 30 x 30 grid, this means the potential for 75,690 parallel operations (Figure 9).

4.2.3. Architectural Goals for the Contour Surface Display Generator

The first goal in the design of the contour surface display generator is to build a system that meets the performance requirements, i.e. a new contour surface display computed from a 30 x 30 x 30 grid, and delivered to a display device in one-thirtieth of a second. This is an ambitious goal but it must be noted that one-thirtieth of a second is the maximum amount of time allowable for the operation. Any longer amount of time does not provide the viewer smooth transitions between successive contour surface displays. This goal says nothing about the load time of the 30 x 30 x 30 grid to the special piece of hardware that computes the contour surface display. Consequently, we allow solutions that pre-load the grid.

The second goal for the construction of the contour surface display generator is the one mentioned above, that we be able to plug it into an existing graphics system with minimal hardware and software changes. For the purposes of this study, the target graphics system is chosen to be the Silicon Graphics, Inc. IRIS workstation [2]. The Silicon Graphics, Inc. IRIS is currently the highest performance graphics system that best matches the selected application's goals.

4.2.4. Architectural Outlines

Given that we have a highly decomposable algorithm for contour surface display generation, and given that our goal is a single board VLSI multiprocessor, there are some simple statements we can make about the system's architecture. The first statement is that it is comprised of an array of independent processors, each processor containing some subpart of the total algorithm (Figure 10). (Note: we call these processors, algorithm component processors.) In the case of the contour surface display generation algorithm, this means that each processor contains one or more 2 x 2 subgrids taken from the larger three-dimensional grid. It also means that each processor is

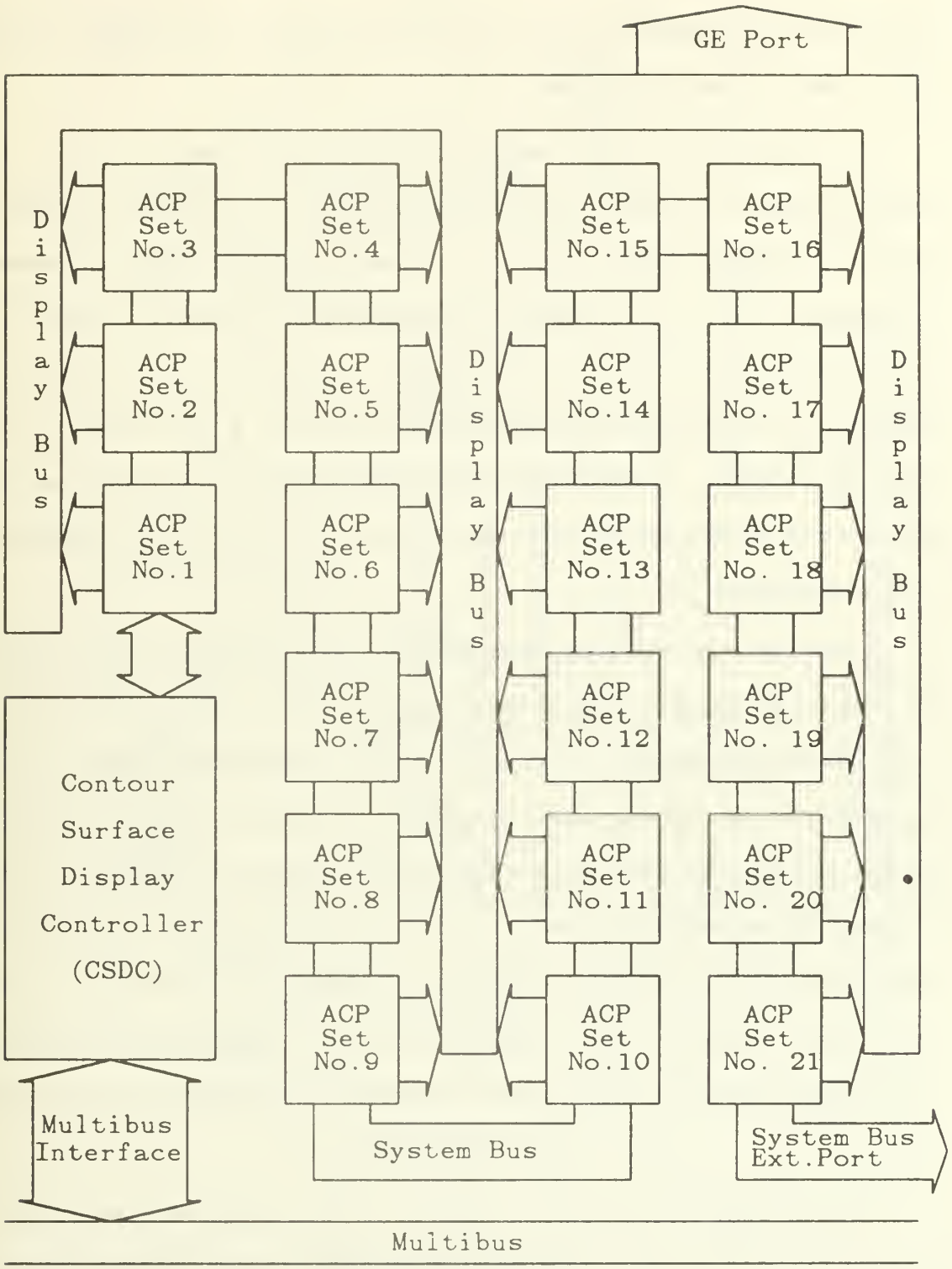


Figure 10 Contour Surface Display Generator.

responsible for computing the part of the surface display represented by its subgrids. The system performs these computations in parallel.

The second statement we can make about the architecture for the contour surface display generator, is that, even though we compute the subparts of the contour surface display generation algorithm in parallel, we need to output the coordinates and drawing instructions generated in each processor in a serial, one processor at a time, fashion. This statement is based upon the requirement that the contour surface display generator be plugged into an existing graphics system (Figure 11). Currently available graphics systems only have a single data path into their display processing units. Consequently, some mechanism needs to be provided to output the data generated from the algorithm component processors, one at a time, to the display processing unit of the graphics system.

A third statement we can make about the architecture is that we need some mechanism for delivering the 2 x 2 subgrids. Subgrid delivery is qualified by the necessity for algorithm component processor addressability, i.e. we need to be able to put each set of subgrids in a predetermined processor. A qualification to this processor addressability capability is that it must be a simple mechanism that doesn't require a large number of control lines and arbitration circuitry. The reason for this qualification is that we expect the addressing mechanism to run between multiple VLSI chips. This qualification is based upon the knowledge that package pins for control lines between VLSI chips are a scarce resource. The output mechanism for the coordinates and drawing instructions of the contour surface display generator needs a similar processor addressability capability.

A fourth statement we can make about the architecture is that we need some mechanism for delivering the new contour levels to the algorithm component processors. The new contour levels can be delivered either in parallel, to the complete system of processors during one cycle, or in serial, to each processor on a separate cycle. Since we are already putting one mechanism in the system for loading data into the processors, we expect that it can also be used to deliver the new contour levels. Consequently, the new contour levels are delivered in serial, in a manner similar

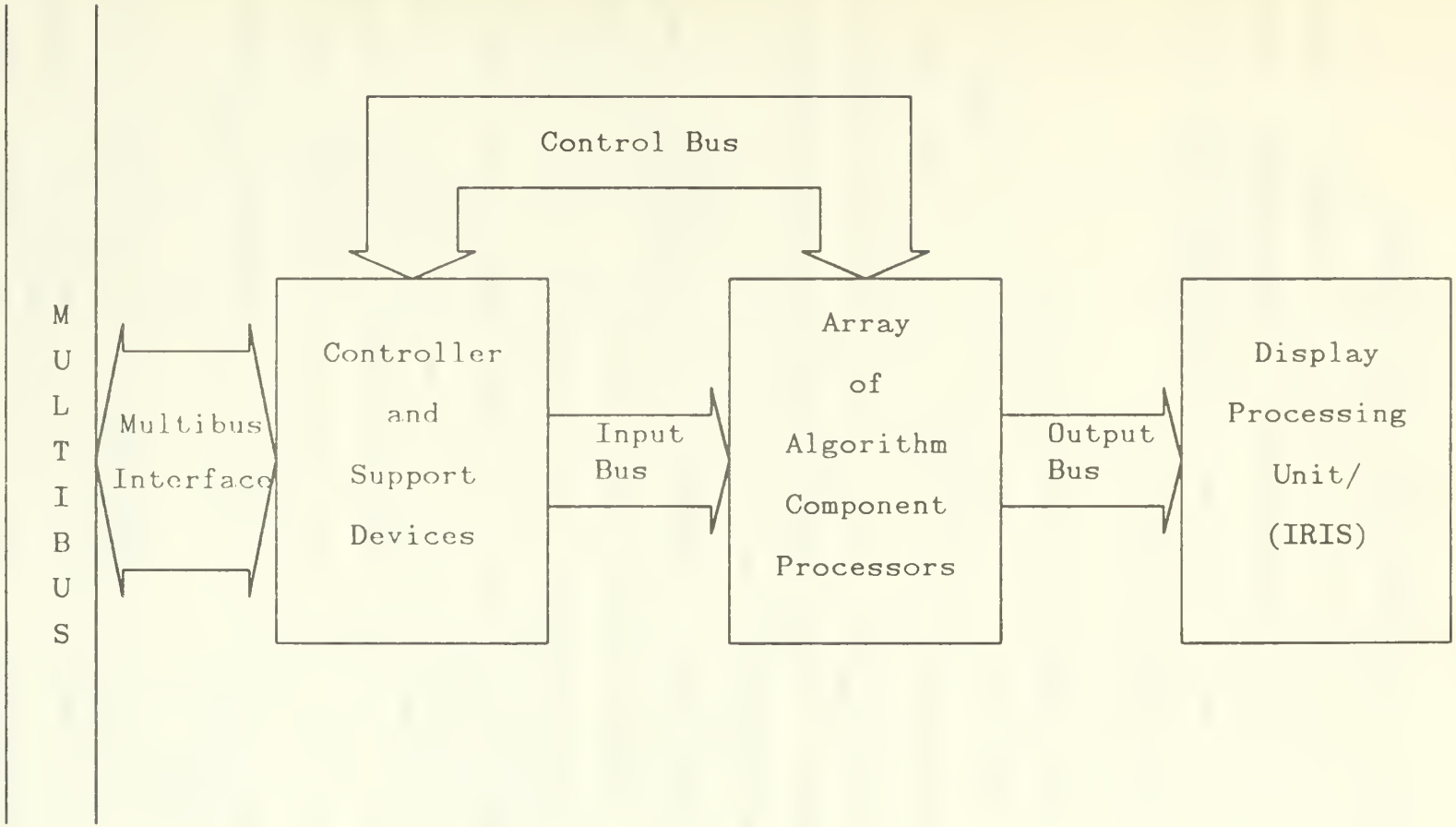


Figure 11
Contour Surface Display Generator: Buses and Data Flow.

to the subgrid load mechanism.

4.2.5. Architecture of the Contour Surface Display Generator

The contour surface display generator is comprised of four subsystems: (1) the array of algorithm component processors, (2) the controller for that array of processors, (3) the algorithm component processor itself, and (4) the interface to the graphics system. Figure 11 shows how the four subsystems relate to the target graphics system.

4.2.5.1. The Array of Algorithm Component Processors

Figure 11 depicts the array of algorithm component processors as a single box, with three connections to the outside environment: an input bus for contour levels and subgrids, an output bus for coordinates and drawing instructions, and a bus for controlling the array of processors. A dual bus configuration is chosen to maximize the amount of concurrency in the system due to the autonomous nature of the input with respect to the output.

The input bus is the medium responsible for delivering subgrid definitions and contour levels to the array of algorithm component processors. Because this is the only data required to be transmitted on the bus, the bandwidth of the input bus does not need to be very high. The rate at which subgrid definitions are loaded into the algorithm component processors does not directly affect the real-time capabilities of the system. The real-time capabilities of the contour surface display generator are determined by the rate at which data can be produced in each algorithm component processor. This, in turn, directly affects the rate of output to the display processing unit. The output bus is responsible for delivering the coordinates and drawing instructions to the display processing unit.

The control bus for the contour surface display generator contains all the control lines necessary to manage the data flow on the input side of the system (Figure 12). Two additional control lines are required on the output side of the system to coordinate the two wire handshake between the algorithm component processors and the display processing unit (Geometry Engines). Figure 12 shows the signals that are needed for all the pin assignments of the algorithm component

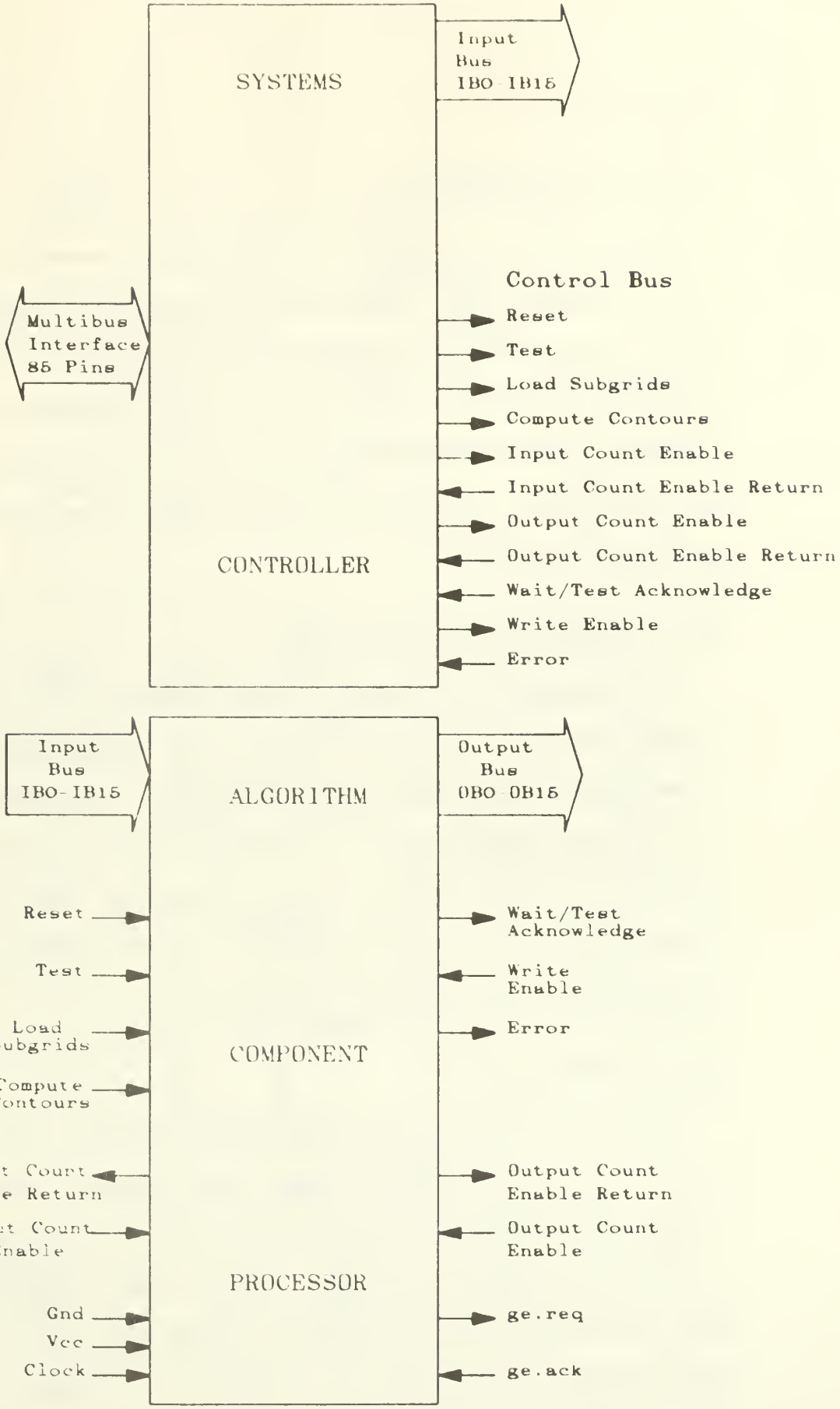


Figure 12
Functional Pin Diagrams of the Systems Controller
and the Algorithm Component Processor

processor.

4.2.5.2. Systems Controller

Control of the array of algorithm component processors involves the integration of several different components. The one which coordinates the operation of all other components is the systems controller. The systems controller converts incoming signals from the Multibus bus master of the Silicon Graphics, Inc. IRIS workstation into signals which make sense to the algorithm component processors. The Multibus bus master is the board in the Multibus Backplane which places the commands on the Multibus. The systems controller is a slave in that it reacts to commands placed on the Multibus.

4.2.5.3. Algorithm Component Processor

The component that is responsible for the production of the coordinates and drawing instructions for the contour surface display generator is the algorithm component processor. Each of these processors is identical and functions independently in the production of the outputs. Figure 13 is an overview diagram of the key components of that processor. We do not go into great detail about that processor other than to point out the items that appear in Figure 13. It should be noted that the processor is a full microprocessor of the Motorola MC68000 class. The reader interested in a more complete treatment is referred to [13].

4.2.5.4. System Interfaces

The contour surface display generator is connected to the Silicon Graphics, Inc. IRIS graphics system by means of the IEEE standard Multibus Backplane Bus [6]. This Multibus connection provides all inputs to the contour surface display generator. The Multibus interfaces to basically two different classifications of bus modules: (1) Masters - those modules which generate commands, and (2) Slaves - those which respond to commands. The parent processor (MC68000) is the Master module for the graphics system. The contour surface display generator is a slave module in that system.

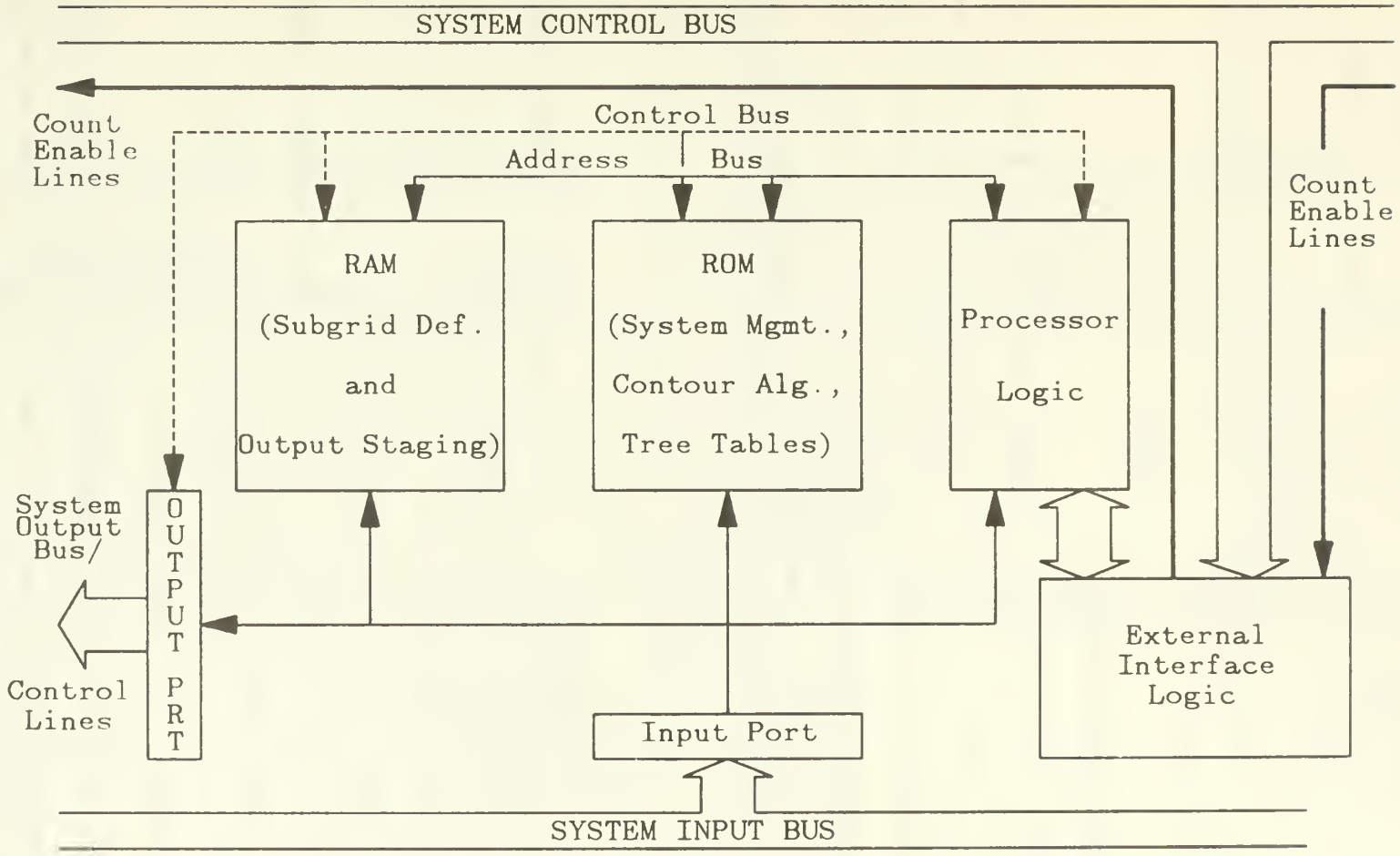


Figure 13
Algorithm Component Processor for the Contour Surface Display Generator.

The output of the contour surface display generator is to the Private Bus of the IRIS system (Figure 11). The Private Bus is a unidirectional, 16 bit bus dedicated to the provision of coordinate and drawing instructions to the high speed Geometry Engines. Coordination of the transfer of data between the algorithm component processors and the Geometry Engines is done via a two line handshake protocol.

4.2.5.5. System Implementation

The IRIS graphics workstation is comprised of the UNIX operating system, the Ethernet communications network and a real-time three-dimensional color-raster graphics system. The hardware elements that make up the workstation are connected to one another via the Multibus (Figure 6). Graphics commands are issued by a host terminal on the workstation. The terminal uses the Motorola MC68000 as a controller and a Geometry Engine pipeline for matrix operations whose output is destined for a high-resolution color raster-scan display. The communications between devices is done on the Multibus. Graphics pipeline data is transferred on the Private Bus.

Graphical output is initiated by the CPU by sending commands and data to the graphics pipeline. The Geometry Engines perform matrix transformations, clipping and scaling. The frame buffer controller interprets characters, controls fonts, and constructs lines and polygons. The update controller does scan conversion of polygons, lines and characters. The results of those operations are placed into the frame buffer. The display controller fetches the picture-element values from the frame buffer and draws them on the face of the color monitor [2].

4.2.5.6. Integration of the Contour Surface Display Generator

The Multibus Backplane of the IRIS supplies the power and inter-board communications capabilities required to implement an integrated graphics system. The contour surface display generator is constructed as a peripheral board and is added to the IRIS graphics system as a slave I/O and memory device. Figure 6 shows the contour surface display generator as an integral part of that system.

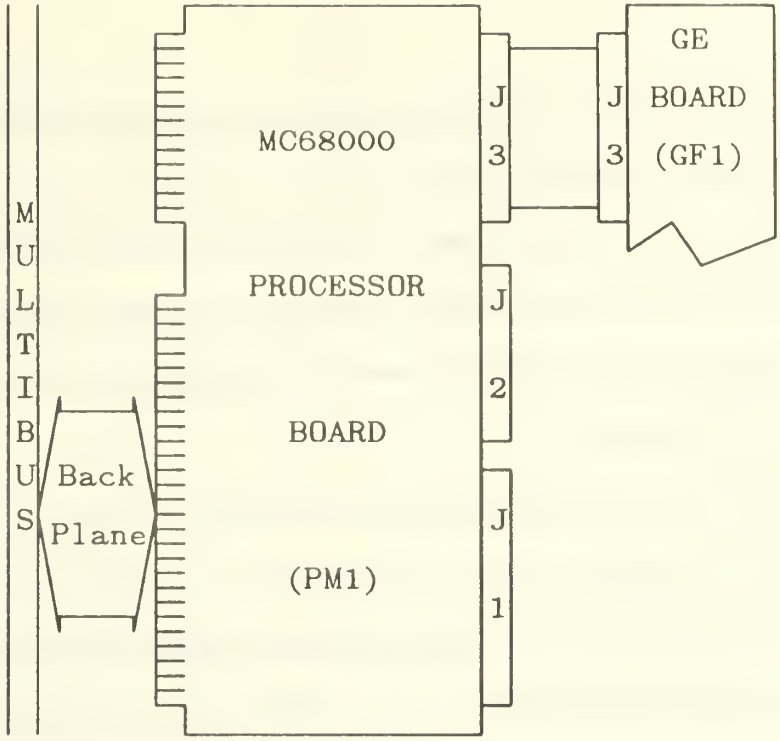


Figure 14a
Silicon Graphics, Inc. IRIS Pipeline Connection
for the Private Bus (Courtesy of Silicon Graphics, Inc.)

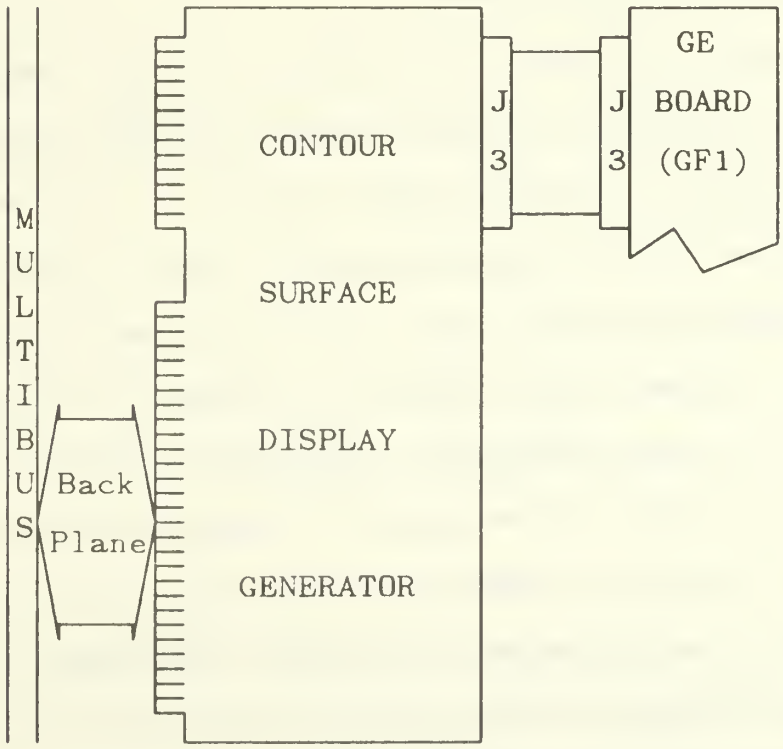


Figure 14b
The J3 Pipeline Connection of the Silicon Graphics, Inc. Private Bus
for the Contour Surface Display Generator

The use of the contour surface display generator in the graphics system involves the establishment of a physical connection between the peripheral board containing the algorithm component processors and the display system. This connection involves the Private Bus port which transports the data directly to the Geometry Engines. In its present configuration, the IRIS system has a connecting cable that directly connects the system processor to the Geometry Engines (J3 connection of Figure 14a).

When the contour surface display generator is added to the system, this physical connection must be shared by both itself and the system processor. To enable the user to alternatively route processor and generator data to the Geometry Engines, a hardware switch is added to the system. This hardware provides the system with a way to multiplex the direct path of the Private Bus. A software switch then provides the control of the Private Bus' origin and configuration. When activated, this switch establishes a path from the contour surface display generator (Figure 14b). If it is not activated, the IRIS system remains in its original configuration.

4.2.6. Hardware Complexity Estimate

The above is a quick overview of the architecture of the contour surface display generator. One of the key components in this system is obviously the algorithm component processor. In 13 , it is determined that 50 algorithm component processors are all that are needed in the system to generate and deliver the average sized picture for a 30 x 30 x 30 grid. In order to determine the feasibility of the complete system, we need a circuit complexity estimate for the size of the algorithm component processor. Figure 15 is a summary of the number of transistor equivalent devices necessary. The derivation of the numbers on that figure is in 13 . We note only that the total number of devices required for one algorithm component processor is about 660K devices. This number is well below the two million devices per chip level that is currently being produced in research laboratories [9]. For this level of chip complexity, the array of algorithm component processors can be built in less than 25 VLSI chips. At the ten million devices per chip level, this is less than 5 VLSI chips. The design of this system is therefore within the grasp of current technology.

(1) RAM space -- (2 devices/bit)		
8192 x 32 bits	=	524,288 devices
(2) ROM space -- (1 device/bit)		
-- Tree tables		
2048 x 16 bits	=	32,768 devices
-- Microcode		
2048 x 32 bits	=	65,768 devices
(3) Processor space --		
-- ALU		
-- Register block		
-- Control section		
-- Data, address and control buses		
-- Refresh logic		
	=	23,000 devices
(4) Interface Unit and Test Bed --		
-- External Interface Logic		
-- Test circuitry		
-- Latches and Drivers		
	=	15,000 devices
Device Total	=	660,582 devices

Figure 15
Algorithm Component Processor's Circuit Complexity Estimate

5. Conclusions

The above is but one example of the fourth cycle of hardware developments for the graphics system. The purpose behind the presentation of this effort is to highlight the possibilities and limitations of research in this cycle. One of the key points described at the start of this paper is that applications users are never satisfied with the graphics capabilities of the currently available system. We need to reexamine this notion in light of the production of the contour surface display generator. We can be assured that once we produce such a system that the applications user will return to us with further demands for either other algorithms, or additional capabilities in the already manufactured system. In order to respond to these desires for special hardware for select applications graphics algorithms, we need to either justify the special hardware effort on the basis of widespread demand, or make the production of that special hardware inexpensive. We cannot count on the widespread demand for any algorithm for which we desire real-time performance. The only solution then is to make the production of that special hardware inexpensive. The first step in that process is to put together a methodology based upon experience with designing such special purpose display generators. Once that methodology is sufficiently developed, we can then set standards for the production of such systems. We can see an analogy in the world of VLSI design. The design and production of special VLSI chips came within the possibilities of the university community after standard interfaces were defined for chip production. Hence, we saw the establishment of "silicon foundries". If we extend this idea to that of the production of special hardware for select graphics algorithms of the applications user, this means that somewhere in the future there will be "real-time, graphics foundries". It is towards this direction that we can expect future developments for workstation graphics capabilities to proceed.

6. References

1. Barry, C. D. and Sucher, J. H. "Interactive Real-Time Contouring of Density Maps," American Crystallographic Association Winter Meeting, Honolulu, March 1979, Poster Session.

2. Clark, J. H. and Davis, T. "Workstation Unites Real-Time Graphics with Unix, Ethernet." *Electronics*, October 20, 1983.

3. Clark, J. H. "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 16, No. 3 (July 1982), p. 127.

4. Frank, Edward H. and Sproull, Robert F. "Testing and Debugging Custom Integrated Circuits," *Computing Surveys*, Vol. 13, No. 4 (December 1981), p. 425.

5. Fuchs, Henry et al. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," Computer Graphics: A Quarterly Report of SIGGRAPH-ACM, Vol. 19, No. 3 (July 1985), p. 111.

6. Intel Corporation Technical Publication, *Intel Multibus Specification*, Intel Corporation, Santa Clara, California, 1982.

7. MacGregor, D., Mothersole, D., and Moyer, B. "The Motorola MC68020," *IEEE Micro*, Vol. 4, No. 4 (August 1984), p. 101.

8. Mead, Carver and Conway, Lynn *Introduction to VLSI Systems*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1980.

9. Micronews "IBM Experimental Million Bit Memory Chip," *IEEE Micro*, Vol. 4, No. 4 (August 1984), p. 119.

10. SIGGRAPH. *VLSI for Computer Graphics*, Special Interest Group on Computer Graphics Annual Meeting, San Francisco, July 1985, Notes for Course 20.

11. Sutherland, I.E. and Mead, C.A. "Microelectronics and Computer Science," *Scientific American*, September 1977, pp. 210-228.

12. Uhr, Leonard *Algorithm-Structured Computer Arrays and Networks*, Orlando, Florida: Academic Press, 1984.

13. Walker, Robert A. and Zyda, Michael J "An Integrated Systems Architecture for Real-Time Contour Surface Display Generation." Technical Report NPS52-85-010, Monterey, California: Department of Computer Science, Naval Postgraduate School, August 1985.

14. Zyda, Michael J. "A Decomposable Algorithm for Contour Surface Display Generation," Technical Report NPS52-84-011, Monterey, California: Department of Computer Science, Naval Postgraduate School, August 1984.

16. Zyda, Michael J. "Joystick Driven Display Rotation and Control Console Management." Technical Memorandum 24, St. Louis: Department of Computer Science, Washington University, November 1980.

Distribution List for Papers Written by Michael J. Zyda

Dr. Henry Fuchs,
208 New West Hall (035A),
University of North Carolina,
Chapel Hill, NC 27514

Dr. Kent R. Wilson,
University of California, San Diego
B-014,
Dept. of Chemistry,
La Jolla, CA 92093

Dr. Guy L. Tribble, III
Apple Computer,
20525 Mariani Ave,
Cupertino, CA 95014

Dr. Victor Lesser,
University of Massachusetts, Amherst
Dept. of Computer and Information Science,
Amherst, MA 01003

Dr. Gunther Schrack,
Dept. of Electrical Engineering,
University of British Columbia,
Vancouver, B.C., Canada V6T 1W5

Dr. R. Daniel Bergeron,
Dept. of Computer Science,
University of New Hampshire,
Durham, NH 03824

Dr. Ed Wegman,
Division Head,
Mathematical Sciences Division,
Office of Naval Research,
800 N. Quincy Street,
Arlington, VA 22217-5000

Dr. Gregory B. Smith,
ATT Information Systems,
190 River Road,
Summit, NJ 07901

Dr. Lynn Conway,
Defense Advanced Research Projects Agency/IPTO,
1400 Wilson Blvd.,
Arlington, VA 22209

Dr. John Lowrance,
SRI International,
333 Ravenswood Ave,
Menlo Park, CA 94025

Dr. David Mizell,
Office of Naval Research,
1030 E. Green St.
Pasadena, CA 91106

Dr. Richard Lau,
Office of Naval Research,
Code 411,
800 N. Quincy St.
Arlington, VA 22217-5000

Dr. Y.S. Wu,
Naval Research Laboratory,
Code 7007,
Washington, D.C. 20375

Dr. Joel Trimble,
Office of Naval Research,
Code 251,
Arlington, VA 22217-5000

Robert A. Ellis,
Calma Company,
527 Lakeside Drive,
Sunnyvale, CA 94086

Dr. James H. Clark,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Shinji Tomita,
Dept. of Information Science,
Kyoto University,
Sakyo-ku, Kyoto, 606, Japan

Hiroshi Hagiwara,
Dept. of Information Science,
Kyoto University,
Sakyo-ku, Kyoto, 606, Japan

Dr. Alain Fournier,
Dept. of Computer Science,
University of Toronto,
Toronto, Ontario, Canada
M5S 1A4

Dr. Andries Van Dam.
Dept. of Computer Science,
Brown University.
Providence, RI 02912

Dr. Brian A. Barsky,
Berkeley Computer Graphics Laboratory,
Computer Sciences Division,
Dept. of Electrical Engineering and Computer Sciences,
University of California,
Berkeley, CA 94720

Dr. Ivan E. Sutherland,
Carnegie Mellon University,
Pittsburg, PA 15213

Dr. Turner Whitted,
208 New West Hall (035A),
University of North Carolina.
Chapel Hill, NC 27514

Dr. Robert B. Grafton,
Office of Naval Research,
Code 433.
Arlington, Virginia 22217-5000

Professor Eihachiro Nakamae,
Electric Machinery Laboratory,
Hiroshima University.
Higashihiroshima 724, Japan

Carl Machover.
Machover Associates,
199 Main Street.
White Plains, New York 10601

Dr. Buddy Dean.
Naval Postgraduate School,
Code 52, Dept. of Computer Science.
Monterey, California 93943

Earl Billingsley.
43 Fort Hill Terrace.
Northhampton, MA 01060

Dr. Jan Cuny.
University of Massachusetts, Amherst
Dept. of Computer and Information Science,
Amherst, MA 01003

Robert Lum.
Silicon Graphics, Inc.
630 Clyde Court.
Mountain View, CA 94043

Jeff Hausch,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Lt. Robert A. Walker,
Naval Sea Systems Command (SEA 61YM),
Department of the Navy,
Washington, DC 20362-5101

Dr. Barry L. Kalman,
Washington University,
Department of Computer Science,
Box 1045,
St. Louis, Missouri 63130

Dr. Wm. Randolph Franklin,
Electrical, Computer, and Systems Engineering Department,
Rensselaer Polytechnic Institute,
Troy, New York 12180-3590

Dr. Gershon Kedem,
Microelectronics Center of North Carolina,
PO Box 12889,
3021 Cornwallis Road,
Research Triangle Park,
North Carolina 27709

Dr. Branko J. Gerovac.
Digital Equipment Corporation.
150 Locke Drive LMO4/H4. Box 1015
Marlboro, Massachusetts 01752-9115

Robert A. Schumacker,
Evans and Sutherland.
PO Box 8700.
580 Arapeen Drive.
Salt Lake City. Utah 84108

R. A. Dammkoehler,
Washington University,
Department of Computer Science.
Box 1045.
St. Louis. Missouri 63130

Dr. Lynn Ten Eyck.
Interface Software.
79521 Highway 99N,
Cottage Grove, Oregon 97424

Toshiaki Yoshinaga,
Hitachi Works, Hitachi Ltd.
1-1. Saiwaicho 3 Chome,
Hitachi-shi, Ibaraki-ken,
317 Japan

Takatoshi Kodaira,
Omika Works, Hitachi Ltd.
2-1, Omika-cho 5-chome,
Hitachi-shi, Ibaraki-ken,
319-12 Japan

Atsushi Suzuki,
Hitachi Engineering, Co. Ltd.
2-1, Saiwai-cho 3-Chome,
Hitachi-shi, Ibaraki-ken,
317 Japan

Toshiro Nishimura,
Hitachi Engineering, Co. Ltd.
2-1, Saiwai-cho 3-Chome,
Hitachi-shi, Ibaraki-ken,
317 Japan

Dr. John Staudhammer,
Dept. of Electrical Engineering,
University of Florida.
Gainesville, Florida 32611

Dr. Lewis E. Hitchner.
Computer and Information Science Dept.
237 Applied Science Building,
University of California at Santa Cruz.
Santa Cruz, California 95064

Dr. Pat Mantey.
Computer Engineering Department,
University of California at Santa Cruz.
Santa Cruz. California 95064

Dr. Walter A. Burkhardt,
University of California, San Diego
Dept. of Computer Science,
La Jolla. California 92093

P. K. Rustagi,
Silicon Graphics. Inc.
630 Clyde Court,
Mountain View, CA 94043

Peter Broadwell,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Norm Miller.
Silicon Graphics, Inc.
630 Clyde Court.
Mountain View, CA 94043

Dr. Tosiyasu L. Kunii,
Department of Information Science,
Faculty of Science,
The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113,
Japan

Dr. Kazuhiro Fuchi,
Institute for New Generation Computer Technology,
Mita-Kokusai Building 21FL.
1-4-28 Mita. Minato-ku, Tokyo 108, Japan

Defense Technical Information Center (2)
Attn: DTIC-DDR
Cameron Station
Alexandria, VA 22314

Library, Code 1424 (2)
Naval Postgraduate School
Monterey, CA 93943

Research Administration Office
Code 012
Naval Postgraduate School
Monterey, CA 93943

Center for Naval Analyses
2000 N. Beauregard Street
Alexandria, VA 22311